SM-KIT

für Commodore-Computer

das Profiwerkzeug für Programmierer



Scherbaumstraße 29
8000 München 83

SOFTWARE-VERBUND MICRO COMPUTER GMBH



SM-KIT

für Commodore-Computer
das Profiwerkzeug für Programmierer

Scherbaumstraße 29
8000 München 83

Telefon: 089/6 37 12 11
Aut. Auftragsannahme
Telefon: 089/6 70 58 13

Druck: Treiber-Offset, München



Inhaltsverzeichnis



Inhaltsverzeichnis

A. VORWORT

Der **SM-KIT** ist eine Sammlung von Programmier- und Testhilfen für BASIC-Programmierer. Es existiert eine **ROM**-Version für **BASIC-4 (80xx/40xx)** und eine **EPROM**-Version für **BASIC-3 (30xx).**

Bei seiner Entwicklung wurden zwei Dinge ganz besonders beachtet: Es sollten keine Scheinhilfsmittel eingebaut werden, nur um mit **'soundsovielen neuen BASIC-Befehlen'** werben zu können und die realisierten Befehle sollten nicht nur gerade noch die Funktion erfüllen, die der Name aussagt, sondern sollten wirklich alle Aspekte der praktischen Arbeit berücksichtigen und möglichst einfach abrufbar sein.

Einer der besten Kenner des Commodore-Betriebssystems, der Autor unserer ROM-Listings, schrieb für Sie den **SM-KIT.** Sie werden seine Liebe zum Detail und seine Abneigung gegen überflüssige Fehlermeldungen sehr bald erkennen und schätzen lernen.

Um die vielfältigen Möglichkeiten des **SM-KIT** problemlos nutzen zu können, bitten wir Sie, diese Anleitung wirklich sorgfältig durchzuarbeiten und anhand der angegebenen oder eigener Beispiele nachzuvollziehen.

Wir haben den **SM-KIT** sehr gewissenhaft getestet und halten ihn für frei von groben Fehlern. Dennoch ist bei einem 4K-Maschinenprogramm, das derart weit und vielfältig ins Betriebssystem eingreifen muß, nicht ausgeschlossen, daß gewisse Nebenwirkungen übersehen wurden. Sollten Sie einen Fehler feststellen, bitten wir Sie, ihn uns schriftlich unter möglichst genauer Darstellung der Vorgeschichte und Begleitumstände mitzuteilen.

Wir wünschen Ihnen viel Spaß und Erfolg mit dem **SM-KIT.**

München im März 1981

SOFTWAREVERBUND MICROCOMPUTER GMBH
H. Schießl

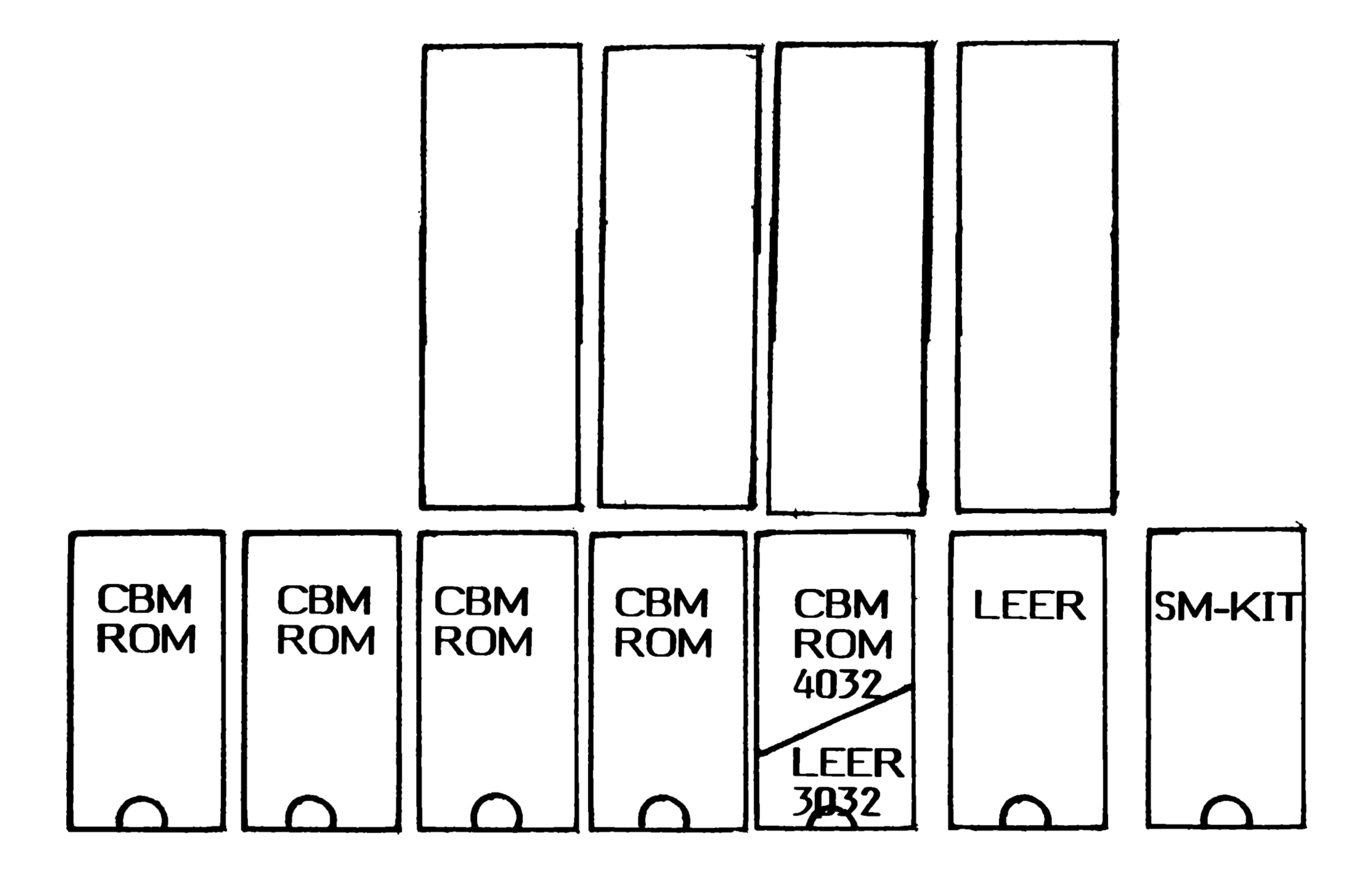
0. EINBAU DES SM-KIT:

Das SM-KIT wird in den letzten freien Sockel eingesetzt. Bitte beachten Sie dabei die Codierung. **Vorsicht** vor elektrostatischer Aufladung!

EINSCHALTEN: SYS36864

AUSSCHALTEN: .X

- 3032/4032





0. EINBAU DES SM-KIT:

- 8032

SM-KIT

LEER

CBM ROM

CBM ROM

CBM ROM

CBM ROM

CBM ROM



1. AUSGABEMÖGLICHKEITEN

SM-KIT bietet die Möglichkeit, die Listen von einigen Befehlen, sowie beliebige Bildschirminhalte auf Drucker oder Floppy auszugeben. Diese Möglichkeiten und die Steuerung der Ausgaben grundsätzlich werden im folgenden beschrieben.

1.1 Steuerung der Ausgabe

Die Befehle **FIND, DUMP, TRACE und DIRECTORY (@$)** geben Listen aus, die länger als ein Bildschirm sein können. Um nicht ungewollt Information durch Hochrollen des Bildschirms zu verlieren, geben diese Befehle die Listen jeweils nur aus, solange die **RETURN**-Taste gedrückt ist. Läßt man los, stoppt die Ausgabe, drückt man wieder, läuft sie weiter.

Um bei langen Listen, die z.B. auf Drucker oder Floppy ausgegeben werden sollen, nicht dauernd die **RETURN**-Taste drücken zu müssen, wird auch weitergelistet, wenn die **SHIFT**-Taste gedrückt wird. Durch Drücken von **SHIFT-LOCK** können Sie also auf Dauerausgabe schalten. Dazu drücken Sie zuerst die **RETURN**-Taste, um die Funktion auszulösen und drücken dann **SHIFT-LOCK,** solange die **RETURN**-Taste noch gedrückt ist. Dann können Sie die **RETURN**-Taste loslassen. Vergessen Sie aber nicht, nach Beendigung der Funktion die **SHIFT-LOCK**-Taste wieder zu lösen.

Die Befehle **FIND, DUMP und DIRECTORY** können durch Drücken der **STOP**-Taste abgebrochen werden, **TRACE** fällt nach Loslassen von **RETURN** automatisch in den Eingabemodus zurück.

1.2 Ausgabe auf Drucker 3022 / 4022

.←´Pfeil nach links´

setzt das Ausgabeflag für die Standarddrucker 3022/4022. Alle SM-KIT Befehle, die Listen erzeugen, geben dann zusätzlich zur Bildschirmausgabe auch auf Drucker aus. Eine Ausnahme bildet **DIRECTORY.**

Da diese Ausgabe speziell auf den 3022/4022 zugeschnitten ist, wird vor jeder Zeile der Code 17 gesendet, um den Drucker auf Kleinschreibung zu schalten, sofern der Rechner ebenfalls auf Kleinschreibung gestellt ist.

Da dieser Steuercode bei anderen Druckern entweder den Druck eines Zeichens bewirken kann, oder den Drucker zu unerwünschten Reaktionen bewegten kann, , sollten Sie für andere Drucker die **´Ausgabe auf beliebiges Peripheriegerät´** verwenden.

.#(Doppelkreuz)

schaltet die Druckerausgabe wieder ab.

1.3 Ausgabe auf beliebiges Peripheriegerät

.←'Pfeil nach links' la

schaltet den Ausgabekanal auf die logische **Adresse la.** Hier ist also vorausgesetzt, daß vor der ersten Ausgabe ein **OPEN** in der üblichen Weise auf das entsprechende Gerät gemacht wurde.

Alle Ausgaben werden dann in analoger Weise zu **PRINT§la** auf dieses Gerät gegeben. Dies bedeutet, daß diese Datei immer offen sein muß, wenn eine Ausgabe gemacht werden soll. Dies ist bei **TRACE** und nach **NUMBER** und **DELETE** zu beachten.

.#(Doppelkreuz) schaltet auch diese Ausgabe ab.

Wurde einmal die **la** hinter **'Pfeil nach links'** angegeben, genügt in Zukunft **'Pfeil nach links'**, um die Ausgabe auf diesen Kanal zu schalten.

Soll auf einen anderen Kanal umgeschaltet werden, braucht nur die entsprechende **la** angegeben werden. Auf den Standardkanal wird durch **la 4** zurückgeschaltet.

Beim Standardkanal braucht keine Datei mit der **la 4** geöffnet sein, es stört aber auch nicht, wenn eine geöffnet ist.

1.4 HARDCOPY

Durch zwei Anweisungen kann ein beliebiger Zeilenbereich des Bildschirms auf Drucker oder Floppy ausgegeben werden.

Für die Ausgabe auf 3022/4022 (Standardkanal) braucht weiter nichts gemacht zu werden, für Ausgabe auf sonstige Geräte muß durch **´Pfeil nach links´la** die entsprechende logische Adresse als Ausgabekanal eingestellt sein.

Die erste und letzte Zeile, die auszugeben ist, wird durch jeweils einen Stern gekennzeichnet. Im einzelnen gehen Sie vor, wie folgt:

Den Cursor in die erste auszugebende Bildschirmzeile bringen und an den Anfang dieser Zeile

.*   (Punkt Stern)

schreiben. In der Regel ist es am einfachsten, wenn Sie dazu vorher mit **INSERT** Platz geschaffen haben. Sobald Sie **RETURN** drücken, wird der Stern durch **DEL** wieder weggelöscht. Wenn nötig, können Sie dann ohne weiteres auch noch den Punkt löschen.

Die letzte auszugebende Zeile markieren Sie in gleicher Weise wie die erste. Beachten Sie aber, daß von **einschließlich** der ersten Zeile (1. Stern) bis **ausschließlich** der letzten Zeile (2. Stern) ausgegeben wird.

Wollen Sie bis zur **letzten** Bildschirmzeile ausgeben, markieren Sie einfach die erste Zeile zweimal.

Bis genau einschließlich der **vorletzten Bildschirmzeile** kann nicht ausgegeben werden, da bei Eingabe des Sterns in der letzten Zeile der Bildschirminhalt hochgeschoben wird.

Die Ausgabe auf Dateien erfolgt ohne führende Anführungszeichen und mit **CHR$(13)** am Ende. Wenn Sie durch SM-KIT erstellte Floppy-Dateien lesen wollen, ist dies in der Regel nur durch **GET** möglich, da **INPUT** bei Komma oder Doppelpunkt die Zeileneingabe beendet.

2. AUTO

SM-KIT stellt bei Eingabe von Programmzeilen automatisch die nächste Zeilennummer zur Verfügung, wenn die Zeilennummer der eingegebenen Zeile in der **ersten Bildschirm-Spalte** beginnt. Die Differenz von der eingegebenen Zeile zur nächsten ist gleich der Differenz zur vorhergehenden Programmzeile.

Einige Beispiele sollen das Verhalten der automatischen **AUTO**-Funktion erklären:

NEW
10 REM ERSTE ZEILE

Nach dieser Zeile wird als nächste Nummer automatisch 20 vorgegeben, da noch keine Zeile im Speicher steht. Nach

20 REM ZWEITE ZEILE

wird 30 vorgegeben usw.

Eine vorgegebene Nummer können Sie jederzeit ändern. Wenn Sie nach dem obigen Beispiel die Nummer 100 eingeben, wird als nächste Nummer 170 vorgegeben. Die 170 ändern Sie einfach in 110 ab und bekommen dann als nächste Nummer 120 vorgegeben.

Nachdem es nicht gut ist, wenn bei Korrekturen an schon bestehenden Zeilen die **AUTO**-Funktion eingeschaltet ist, ist die denkende **AUTO**-Funktion im Listing abgeschaltet. Der Zusammenhang ist ganz einfach: **LIST** druckt die Zeilennummern immer ab der 2. Spalte. Natürlich können Sie auch willkürlich in der zweiten Spalte zu schreiben beginnen und dadurch **AUTO** abschalten. Aber **AUTO** denkt auch dann für Sie mit und setzt in der nächsten Zeile den Cursor in der zweiten Spalte auf.

Durch einfaches Wechseln von der ersten in die zweite Spalte und umgekehrt können Sie also **AUTO** ein- oder ausschalten.

3. DELETE

Durch **DELETE** können Sie beliebige Zeilenbereiche eines Programms im Arbeitsspeicher löschen. Der Zeilenbereich wird dabei in derselben Form wie hinter **LIST** angegeben.

Nach **DELETE** ist das Programm genauso kalt, wie wenn Sie von Hand eine Zeile gelöscht hätten, es wird also automatisch die Funktion **CLR** ausgeführt.

Beispiel:

.D100-500	Zeilen 100 bis 500 löschen
.D-500	Alle Zeilen bis 500 löschen
.D100-	Alle Zeilen ab 100 löschen

4. NEU-NUMMERIEREN (NUMBER)

Mit **.N** kann jeder Teil eines Programmes andere Zeilen-Nummern erhalten. Falls nötig, wird der umnummerierte Zeilenbereich an eine andere Stelle des Programmes transportiert. Durch **.N** kann also gleichzeitig die Blockreihenfolge eines Programms geändert werden.

Eine Referenzliste von alten zu neuen Zeilennummern kann auf Drucker ausgegeben werden und erleichtert so die Dokumentation.

4.1 Format

.N quellzeilenbereich **, erste zielzeile , schrittweite**

quellzeilenbereich wird im **LIST**-Format angegeben und kann auch ganz entfallen. Wird der Quellenzeilenbereich nicht angegeben, so wird das ganze Programm nummeriert.

erste zielzeile gibt an, ab welcher Zeilennummer die Zeilen des Quellbereichs abgelegt werden sollen. Dieser Parameter muß angegeben werden.

schrittweite gibt an, in welchen Abständen die neuen Nummern vergeben werden sollen. Hier sind Werte von 1 bis 255 zugelassen. Die Schrittweite muß angegeben werden.

Beispiele

.N 100-500,100,10	Bereich von 100 bis 500 in 10-er Schritten nummerieren
.N 100-,100,5	Bereich ab 100 in 5-er Schritten nummerieren

.N -500,1,1	Bereich bis 500 in 1-er Schritten nummerieren
.N ,10,10	Ganzes Programm ab 10 in 10-er Schritten nummerieren
.N 100-500,1000,5	100 - 500 ab 1000 in Schritten von 5 ablegen
.N 100-500,10,10	100 - 500 ab 10 in Schritten von 10 ablegen

4.2 Fehlermeldungen

? wird gemeldet, wenn der Quellzeilenbereich leer ist.

OVERFLOW ERROR wird gemeldet, wenn im Zielbereich eine alte Zeile stehen würde, also eine Zeile, die nicht durch **.N** dorthin gebracht wurde, oder wenn durch eine neue Zeilennummer eine schon vorhandene Zeile gelöscht werden würde. Bei **OVERFLOW ERROR** wird **.N** nicht ausgeführt.

65535 als Zielzeilennummer wird vergeben, wenn ein Sprungziel nicht definiert ist, wenn also ein Sprung einen **UNDEF'D STATEMENT ERROR** ergeben würde. Läuft allerdings später das Programm oder auch **.N** auf eine solche Zeilennummer **65535**, so wird **SYNTAX ERROR** gemeldet.

Wir empfehlen deshalb nach **.N** durch **.F,65535** nach dieser Nummer zu suchen, um Überraschungen zu vermeiden.

SYNTAX ERROR erscheint, wenn im Programm eine Zeilennummer vorkommt, die größer als **63999** ist. Dies ist immer dann der Fall, wenn ein Programm schon durch **.N** behandelt wurde und dabei eine nicht vorhandene Zielzeile gefunden wurde. Solche Zeilen werden in **65535** umgewandelt, welche dann beim nächsten Lauf zu dieser Fehlermeldung führen.

Diese Fehlermeldung tritt erst auf, nachdem die Referenztabelle gedruckt wurde. Verwechseln Sie diese Meldung nicht mit einem **SYNTAX ERROR** der sofort nach der Eingabe von **.N** auftritt und ein falsches Kommando-Format meldet.

Wenn diese Meldung wegen 65535 auftritt, ist das BASIC-Programm zerstört, weil es nur teilweise umnummeriert ist. Suchen Sie also vor **.N** immer mit **FIND** nach 65535 oder speichern Sie das Programm **vor .N** ab!

ILLEGAL QUANTITY ERROR tritt auf, wenn durch **.N** der zulässige Zeilennummerbereich überschritten werden würde, wenn also Zeilennummern größer als **63999** erzeugt werden müßten. **.N** wird in diesem Fall nicht ausgeführt.

OUT OF MEMORY ERROR tritt auf, wenn der Speicherplatz für die Referenztabelle nicht ausreicht. Die Tabelle benötigt pro umzunummerierender Zeile 4 Bytes. **.N** wird in diesem Fall nicht ausgeführt.

Beispiele:

LIST

```
10   GOTO20
20   GOTO15
30   GOTO10
READY.
```

Zeile 20 würde **Undef'd Statement Error** hervorrufen

Beispiele:

.N,100,10

```
21449 BYTES FREE
10    100
20    110
30    120
```

LIST

```
100   GOTO 110
110   GOTO 65535
120   GOTO 100
READY.
```

undefinierte Zielzeile wurde in 65535 umgewandelt

.N,10,10

```
21444 BYTES FREE
100    10
110    20
120    30
```

?SYNTAX ERROR
READY.

wegen 65535 in Zeile 10

LIST

```
100    GOTO 20
110    GOTO 65535
120    GOTO 100
```

LIST

```
100    GOTO 120
120    GOTO 100
200    GOTO 100
READY.
```

.N100-199,200,10

?OVERFLOW ERROR
READY.

weil die alte Zeile 200 durch die alte Zeile 100 überschrieben werden würde

.N100-199,100,100

?OVERFLOW ERROR
READY

weil die alte Zeile 200 durch die alte Zeile 120 überschrieben werden würde.

.N100-199,190,20

?OVERFLOW ERROR
READY.

weil die alte Zeile 200 im neuen Zielbereich liegen würde, wodurch Zeilenblöcke durcheinandergewürfelt würden.

4.3 Anmerkungen

.N beinhaltet die Funktion **CLR.**

Bei großen Programmen kann die **Zeit** für **.N** einige Minuten betragen. Beachten Sie dabei, daß die eigentlichen Korrekturen am Programm erst vorgenommen werden, nachdem die Referenztabelle gedruckt wurde.

.N hat das Programm erst dann vollständig umnummeriert, wenn der blinkende Cursor wieder erscheint. **.N** ist durch **STOP** unterbrechbar, in diesem Fall ist das Programm aber unbrauchbar. Drücken Sie **STOP** während **.N** ,die Referenztabelle druckt, wird nach der Tabelle abgebrochen, und das Programm ist noch unverändert. Diese Eigenschaft können Sie ausnützen, um die Zeilenanzahl Ihres Programmes zu bestimmen.

Wollen Sie die Referenzliste auf Drucker ausgeben, müssen Sie vorher die Ausgabe auf Drucker durch **.←"Pfeil nach links"** einschalten.

4.4 Beispiel für Blocktransport

Der folgende Nonsens soll ein Programm darstellen, das aus drei Blöcken besteht, wobei Sprungverbindungen zwischen allen Blöcken existieren.

```
10 REM BEREICH1
20 GOTO 120
30 GOSUB130
40 IF A = B THEN140
50 ON I GOTO 50,150,1150
110 REM BEREICH2
120 GOTO 1120
130 GOSUB1130
140 IF A = B THEN1140
150 ON I GOSUB50,150,1150
1110 REM BEREICH3
1120 GOTO 20
1130 GOSUB 30
1140 IF A = B THEN40
1150 ON I GOTO 50,150,1150
```

Durch die folgende Anweisung soll der erste Block mit Zeilennummern ab 10000 versehen werden. Da diese Nummern größer sind, als die Nummern der beiden folgenden Blöcke, muß Block 1 am Anfang des Programms gelöscht werden und stattdessen am Ende wieder ´eingepflanzt´ werden.



Beispiele:

.N-99,10000,100

Nach Eingabe dieser Anweisung wird die Referenztabelle ausgedruckt. Falls Sie durch **.'Pfeil nach links'** die Ausgabe auf Drucker eingeschaltet haben, wird die Tabelle auf Drucker ausgegeben.

```
10 10000
20 10100
30 10200
40 10300
50 10400
```

LIST

```
110 REM BEREICH2
120 GOTO 1120
130 GOSUB1130
140 IF A = B THEN1140
150 ON I GOSUB10400,150,1150
1110 REM BEREICH3
1120 GOTO 10100
1130 GOSUB10200
1140 IF A = B THEN10300
1150 ON I GOTO 10400,150,1150
10000 REM BEREICH1
10100 GOTO 120
10200 GOSUB130
10300 IF A = B THEN140
10400 ON I GOTO 10400,150,1150
READY.
```

Mit Ausnahme der Überlappungsfälle, die durch **OVERFLOW ERROR** angezeigt und verhindert werden, können Sie ein Programm durch **.N** umstrukturieren, ohne seine Funktion zu verändern, solange Sie dadurch icht Programmblöcke trennen.



5. HELP

SM-KIT druckt automatisch bei Fehlermeldungen die falsche Zeile aus und setzt den Cursor an der Stelle auf, wo der Interpreter ausgestiegen ist. Der Fehler liegt dann natürlich nicht unbedingt ganz exakt an dieser Stelle, aber zwei wertvolle Hinweise gibt die Cursorstellung doch. Der Fehler kann ganz sicher nicht weiter rechts sein, als der Cursor steht, sondern er muß sicher links vom Cursor liegen, bzw. genau an der Cursorstelle. Und wenn vor der Anweisung, in der der Cursor steht, noch eine Anweisung kommt, kann der Fehler auch nicht in der vorhergehenden Anweisung liegen.

Dies bedeutet, daß der Fehler genau in der Anweisung aufgetreten ist, in der der Cursor steht. Damit ist es in der Regel sehr einfach, ihn zu lokalisieren. Zusätzlich können Sie Hilfsmittel wie **DUMP** oder **PRINT** einsetzen, um den Fehler noch genauer einzukreisen.

HELP ist während **TRACE** abgeschaltet.



6. FINDE

Mit **.F** kann das Programm nach bestimmten Anweisungsteilen oder Texten durchsucht werden. Bei Treffern wird die gefundene Zeile komplett ausgelistet. Die gesamte Liste kann auf Drucker ausgegeben werden.

6.1 Format

.F zeilenbereich **,gesuchte anweisung**

zeilenbereich wird im **LIST**-Format angegeben und kann auch ganz entfallen. Wenn kein Zeilenbereich angegeben wird, wird das ganze Programm durchsucht. Das Komma, das den Zeilenbereich von der gesuchten Anweisung trennt, muß in jedem Fall eingegeben werden.

gesuchte anweisung darf einschließlich Komma und Anführungszeichen alle BASIC-Zeichen enthalten. Beachten Sie, daß die gesuchte Anweisung unmittelbar nach dem Komma beginnt, daß hier also ausnahmsweise Blanks von Bedeutung sind!

Suchen Sie Text, so geben Sie nach dem Komma noch einen Apostroph ein. Der angegebene Text wird dann nur hinter Anführungszeichen oder REM gesucht.

Beispiele

.F10-99,A$=
Zeilen 10-99 nach **A$=** durchsuchen

.F,FOR
ganzes Programm nach **FOR** durchsuchen

.F-99,´ABCD
bis 99 nach dem Text **ABCD** durchsuchen

.F,"
ganzes Programm nach **"** durchsuchen

.F,$
alle einfachen Stringvariablen

.F,$(
alle indizierten Stringvariablen

.F,%
alle einfachen Integervariablen

.F,%(
alle indizierten Integervariablen

6.2 Anmerkungen

Die Ausgabe auf Bildschirm oder Drucker läuft nur, solange die **RETURN**-Taste gedrückt wird. Durch Loslassen und Drücken von **RETURN** kann also die Ausgabe bequem gesteuert werden. Durch Drücken der **STOP**-Taste wird die Funktion ganz abgebrochen. Durch **SHIFT-LOCK** kann die Ausgabe auf Dauer geschaltet werden.

Angegebene Variable werden eindeutig gefunden, d.h. bei **.F,A** wird nicht auch **AB** oder **A$** gemeldet.

Zahlen werden ebenfalls eindeutig gefunden. Bei **.F,5** wird nicht auch **GOTO 50** gemeldet. Durch einen **Ziffern-joker (!)** können aber Zahlen mehrdeutig gesucht werden. ! kann für jede Ziffer stehen, allerdings nicht als erste Ziffer einer Zahl.

Soll die Liste der Treffer-Zeilen auf Drucker oder Floppy-Datei ausgegeben werden, so muß vorher durch **.´Pfeil nach links´** die Ausgabe eingeschaltet werden.

BASIC-Anweisungen werden vor der Suche übersetzt in die 1-Byte-Interpreterabkürzungen. Bei der Auswahl der zu suchenden Zeichen muß also die Zuordnung von Abkürzungen zu **LIST**-Worten bekannt sein. Vor allem bei Doppelkreuz, Dollarzeichen und Klammern können Probleme entstehen.

INPUT# und PRINT# wird mit dem Doppelkreuz als ein Byte abgespeichert. Deshalb findet ein Doppelkreuz alleine oder **INPUT und PRINT** alleine nicht diese beiden Anweisungen. Dagegen wird **GET#** als zwei Bytes abgelegt, so daß sowohl **GET** als auch Doppelkreuz alleine **GET#** findet.

Die Stringfunktionen **STR$, CHR$, LEFT$, RIGHT$ und MID$** können ebenfalls nicht durch **$** alleine gefunden werden. Außerdem würde **$** alle einfachen Stringvariablen finden.

Bei **TAB(und SPC(** ist zu beachten, daß bei diesen beiden Funktionen die Klammer mit in der Abkürzung enthalten ist. Deshalb findet weder **TAB oder SPC,** noch die Klammer diese beiden Funktionen.

6.3 Beispiele

Ehe die einzelnen **FIND**-Beispiele erläutert werden, zeigen wir zuerst unser Testprogramm, auf das die Beispiel-FINDs losgelassen werden. Sowohl das ganze Programm als auch manche Anweisungen sind völlig sinnlos. Sie sollen lediglich zeigen, was gefunden wird und was nicht.



LIST

```
    10 REM EINFACHE VARIABLE
    20 A = 1
    30 AB = 2
    40 A% = 3
    50 AB% = 4
    60 A$ = "5"
    70 AB$ = "6"
    100 REM INDIZIERTE VARIABLE
    110 A(1) = 1
    120 AB(2) = 2
    130 A%(3) = 3
    140 AB%(4) = 4
    150 A$(5) = "5"
    160 AB$(6) = "6"
    200 REM ZAHLEN/ZIFFERN/ZIFFERNJOKER
    210 GOTO 5
    220 ON I GOTO 70,50
    230 GOTO 535
    300 REM STRINGS, REMS
    310 C$ = "123456"
    320 D$ = ""
    330 D$ = "ABC"+"CDE":REMEFG
    400 REM KOMMA,DOPPELPUNKT
    410 A$ = MID$(A$,B)
    420 A$ = MID$(A$,C)
    425 PRINTMID$("AF",1)
    430 A$ = B$:C$ = E$
    500 REM § / $ / (
    510 PRINT§4
    520 C$ = MID$(A$,B)
    530 A = ASC("3")
    540 PRINTTAB(10)
READY.
```



Die folgenden Beispiele zeigen, wie Variable gesucht und gefunden werden. Wenn Sie jeweils mit dem Programm vergleichen, sehen Sie, daß Variable ganz exakt gefunden werden.

.F-199,A 20 A=1	einfache Gleitkommavariable A
.F-199,AB 30 AB=2	einfache Gleitkommavariable AB
.F-199,A% 40 A%=3	einfache Integervariable A%
.F-199,AB% 50 AB%=4	einfache Integervariable AB%
**.F-199,A$** 60 A$="5"	einfache Stringvariable A$
**.F-199,AB$** 70 AB$="6"	einfache Stringvariable AB$
.F-199,A(110 A(1)=1	indizierte Gleitkommavariable A(
.F-199,AB(120 AB(2)=2	indizierte Gleitkommavariable AB(
.F,A%(130 A%(3)=3	indizierte Integervariable A%(
.F,AB%(140 AB%(4)=4	indizierte Integervariable AB%(
**.F,A$(** 150 A$(5)="5"	indizierte Stringvariable A$(
**.F,AB$(** 160 AB$(6)="6"	indizierte Stringvariable AB$(

Die folgenden vier Beispiele zeigen vier Sonderfunktionen zur Variablensuche:

.F,% alle einfachen Integervariablen

```
40 A%=3
50 AB%=4
```

.F-199,$ alle einfachen Stringvariablen

```
60 A$="5"
70 AB$="6"
```

.F,%(alle indizierten Integervariablen

```
130 A%(3)=3
140 AB%(4)=4
```

.F,$(alle indizierten Stringvariablen

```
150 A$(5)="5"
160 AB$(6)="6"
```

Die folgenden Beispiele demonstrieren die Möglichkeiten der Zahlensuche mit und ohne Ziffernjoker:

```
.F,5                        die Zahl 5 suchen
   150 A$(5)="5"            diese Zeile wird nicht
   210 GOTO5                wegen "5" gefunden (Text)

.F,!5                       Joker an erster Stelle
                            nicht möglich

.F,5!
   220 ON I GOTO 70,50           ersetzt an zweiter
                                 Stelle jede Ziffer

.F,5!!                      dreistellige Zahl mit 5 als
   230 GOTO535              erster Ziffer wird gesucht

.F,5!5                      dreistellige Zahl mit 5 als ersten
   230 GOTO535              und letzten Ziffern werden gesucht

.F,5!6                      dreistellige Zahl mit 5 als ersten
   231 GOTO526              und 6 als letzten Ziffern werden
                            gesucht
```

Im Folgenden wird Text gesucht. Beachten Sie, daß Text (nach Apostroph) nur hinter Anführungszeichen oder REMs gesucht wird.

Text wird auch als Teilstring gefunden:

```
.F,´12
   310 C$="123456"

.F,´BC
   330 D$="ABC"+"CDE":REMEFG

.F,´FG
   330 D$="ABC"+"CDE":REMEFG
```

Die Suche nach **REMs** schafft schnellen Überblick:

```
.F,REM
   10 REM EINFACHE VARIABLE
   100 REM INDIZIERTE VARIABLE
   200 REM ZAHLEN/ZIFFERN/ZIFFERNJOKER
   300 REM STRINGS, REMS
   330 D$ ="ABC" + "CDE":REMEFG
   400 REM KOMMA,DOPPELPUNKT
   500 REM § / $ / (
```

Kommas im Suchstring werden verstanden und gefunden

```
.F,A$,B
   410 A$ = MID$(A$,B)
   520 C$ = MID$(A$,B)
```

Doppelpunkt im Suchstring ist erlaubt

```
.F,B$:C$
   430 A$ = B$:C$=E$
```

Die folgenden Beispiele weisen auf Probleme hin, die durch die Übersetzung von BASIC-Befehlsworten in ein Byte entstehen.

Die Befehle **LEFT$, RIGHT$ und MID$** werden zusammen mit dem Dollarzeichen als ein Byte abgelegt. Deshalb kann man durch Suche nach einem Dollarzeichen diese Befehle nicht finden.

.F421-429,$

Dollarzeichen findet nicht MID$
(würde alle einfachen Stringvariablen finden)

.F421-429,MID MID findet nicht MID$

**.F421-429,MID$** MID$ findet MID$

```
425 PRINTMID$("AF",1)
```

.F421-429,$(

Auch $ mit Klammer findet nicht MID$ (würde alle indizierten Stringvariablen finden)

.F,(

```
110 A(1) = 1
120 AB(2) = 2
130 A%(3) = 3
140 AB%(4) = 4
150 A$(5) = "5"
160 AB$(6) = "6"
410 A$ = MID$(A$,B)
420 A$ = MID$(A$,C)
425 PRINTMID$("AF",1)
520 C$ = MID$(A$,B)
530 A = ASC("3")
```

geöffnete Klammer findet sowohl alle indizierten Variablen als auch alle Funktionen und sonstigen Klammern

.F,# Doppelkreuz alleine findet nicht **INPUT#oder PRINT#, aber GET#.**

.F,PRINT# findet **PRINT#,** aber nicht **PRINT**

```
510 PRINT#4
```

Bei **TAB(und SPC(** gehört die Klammer zum Code. Deshalb werden diese beiden Codes nicht durch eine separate Klammer gefunden. Aus dem gleichen Grund findet **TAB** oder **SPC** nicht **TAB(** oder **SPC(!**

```
.F421-,(
   425 PRINTMID$("AF",1)
   520 C$ = MID$(A$,B)
   530 A = ASC("3")

.F,TAB

.F,TAB(
   540 PRINTTAB(10)
```

7. VARIABLEN-DUMP

Mit **.V** können die Inhalte der einzelnen Variablenarten ausgedruckt werden. Dabei wird unterschieden zwischen einfachen und indizierten Variablen, sowie zwischen den drei Typen bei den einfachen Variablen. Bei den indizierten kann sowohl die Dimensionierung aller Variablen als auch der Inhalt aller Elemente einer bestimmten Variable ausgegeben werden.

7.1 Format

.V variablenart

.V	alle einfachen Gleitkommavariablen mit Inhalt
.V%	alle einfachen Integervariablen mit Inhalt
.V$	alle einfachen Stringvariablen mit Inhalt
.V,	Dimensionierung aller indizierten Variablen (keine Inhalte)
.V,A	Inhalte aller Elemente der indizierten Variablen A
.V,A(3)	Inhalte aller Elemente der indizierten Variablen A ab dem Element 3

7.2 Anmerkungen

Soll die Ausgabe auch auf Drucker oder Floppy gehen, so muß vorher durch **.'Pfeil nach links'** die Ausgabe eingeschaltet werden.

Die Ausgabe läuft nur, solange die **RETURN**-Taste gedrückt wird oder wenn **SHIFT-LOCK** gedrückt ist. Durch Drücken der **STOP** -Taste kann die Funktion abgebrochen werden.



Bei den indizierten Variablen wird die Ausgabe unterbrochen, sobald einmal alle Elemente ausgegeben wurden, sobald also wieder das Element 0 in jeder Dimension erreicht ist.

Wenn Sie mit dem Cursor in die Liste gehen, die nach V, ausgedruckt wird, können Sie durch **2*RETURN** die Ausgabe der Inhalte der betreffenden Variable erreichen, ohne nochmal die Variable mit allen Indizes tippen zu müssen.

Bitte beachten Sie, daß **DUMP** nur die Variablen ausgeben kann, die bis zu diesem Zeitpunkt im Programm angetroffen wurden. Die Variablen werden in der Reihenfolge ausgegeben, wie sie in den Variablenlisten stehen, also die im Programm zuerst angetroffenen zuerst.

7.3 Beispiel

LIST

```
10 A = 1:B = 2
20 A% = 10:B% = 20
30 A$ = "AFADGDAF":B$ = "ADFADAGDF"
40 D1 = 3:D2 = 2
50 DIMA(D1,D2),A%(D1,D2),A$(D1,D2)
60 FOR I1 = 0 TO D1
70 FORI2 = 0 TO D2
90 A(I1,I2) = I1+I2
100 A%(I1,I2) = I1+I2
110 A$(I1,I2) = STR$(I1)+STR$(I2)
120 NEXT:NEXT
READY.
```

RUN

READY.

.V

```
A  = 1
B  = 2
D1 = 3
D2 = 2
I1 = 4
I2 = 3
```

.V%

```
A% = 10
B% = 20
```

.V$

```
A$ = AFADGDAF
B$ = ADFADAGDF
```

.V,
.V,A(3,2)
.V,A%(3,2)
.V,A$(3,2)

Diese Zeile kann durch Cursorbewegungen und **RETURN** übernommen werden, anstatt Sie nochmal einzugeben. **RETURN** in dieser Zeile startet die Ausgabe.

```
.V,A$(3,2)    =  3 2

.V,A$(0,0)    =  0 0

   A$(1,0)    =  1 0
   A$(2,0)    =  2 0
   A$(3,0)    =  3 0
   A$(0,1)    =  0 1
   A$(1,1)    =  1 1
   A$(2,1)    =  2 1
   A$(3,1)    =  3 1
   A$(0,2)    =  0 2
   A$(1,2)    =  1 2
   A$(2,2)    =  2 2
   A$(3,2)    =  3 2
V,A$(0,0)
```

Nach einem vollständigen Durchlauf wird automatisch gestoppt

Bei Vorgabe von Indizes wird ab diesen bis zum Ende ausgegeben

```
V,A$(1,1)    =  1 1

   A$(2,1)    =  2 1
   A$(3,1)    =  3 1
   A$(0,2)    =  0 2
   A$(1,2)    =  1 2
   A$(2,2)    =  2 2
   A$(3,2)    =  3 2
V,A$(0,0)
```

8. PROGRAMM-TRACE

Mit **.R oder .G** kann der Ablauf eines Programms auf dem Bildschirm und auf Drucker oder Floppy protokolliert werden. Zusätzlich zu den durchlaufenen Anweisungen können die Werte mehrerer Variablen ausgegeben werden. Der Aufsetzpunkt im Programm und der Beginn des Trace können frei vorgegeben werden. Die protokollierten Variablen können jederzeit geändert werden.

8.1 Format

.R startzeile **Kaltstart (RUN)**
.G startzeile **Warmstart (GOTO)**

.R entspricht dem Start mit **RUN,** es werden also alle Variablen gelöscht und alle Dateien geschlossen.

Wird hinter **.R keine Zeilennummer** angegeben, wird das Programm bei der ersten Zeile gestartet **(RUN).** Wird dagegen eine Zeilennummer angegeben (.R55), so wird das Programm bei dieser Zeile gestartet (RUN55).

Wird **.G ohne Zeilennummer** verwendet, so hat es die Wirkung eines **CONT** und kann dementsprechend erst angewendet werden, wenn das Programm schon durch **.R** oder durch **.G mit Zeilennummer** gestartet wurde.

.G mit Zeilennummer (.G55) entspricht **GOTO (GOTO55).**

.G -tracebeginn

Der Tracebeginn muß nicht unbedingt mit der Startzeile zusammenfallen. Folgende Möglichkeiten bieten sich für **.G und .R:**

.G -500	Programm ab der derzeitigen Stelle frei weiterlaufen lassen **(CONT)** und Trace ab 500 einschalten.
.G 20-500	Programm mit **GOTO20** starten und Trace ab 500 einschalten.

Wenn Sie nach diesen Anweisungen die **RETURN** -Taste sofort wieder loslassen, stoppt das Programm automatisch nach Erreichen der Zeile, in der der Trace beginnen soll.

Der Trace setzt nur ein, wenn die Zeile tracebeginn wirklich ausgeführt wird. Es erfolgt also nicht etwa eine Abfrage auf kleiner oder größer, sondern nur auf gleich.

Wird tracebeginn angegeben, wird eine eventuell angegebene Variablenliste ignoriert.

.G, variablenliste

Hinter dem Komma nach ´**startzeile**´ können mehrere Variablen oder Ausdrücke angegeben werden. Die Inhalte der Variablen bzw. die Ergebnisse der Ausdrücke werden bei der Ausführung jedes Statements ausgegeben.

Die einzelnen Variablen oder Ausdrücke werden jeweils durch Komma getrennt.

Beispiel:

.G,A$,B,C%,D(I),D(I+1),D(3),3*A,CHR$(A)

8.2 Indizierte Variable

Die Variablenliste wird im Prinzip abgearbeitet wie bei **PRINT.** Deshalb sind insbesondere bei der Indexangabe numerische Ausdrücke erlaubt. Beachten Sie aber hier zwei Fehlermöglichkeiten, die sich aus den Eigenschaften des Interpreters ergeben:

Wenn im Programm indizierte Variable dimensioniert **(DIM)** werden, dürfen sie vorher nicht in der Trace-Variablentabelle vorkommen, da sonst bei der Dimensionierung ein **REDIM'D ARRAY ERROR** auftritt.

Beispiel:

```
10 DIM A(20)
20 FOR I = 1 TO 20 : A(I) = I*2 : NEXT
30 REM
```

.R,A(1)

Diese **TRACE**-Anweisung ergibt **REDIM'D ARRAY ERROR IN 10.**

Sie umgehen diesen Fehler durch die beiden Anweisungen

.R-20

```
20 FOR I = 1 TO 20
```

.G,A(1)

Der zweite Fehler tritt auf, wenn Feldelemente abhängig von der Laufvariablen ausgegeben werden und die Schleife verlassen wird:



Beispiel

.R-20
.G,A(I)

Wird diese Anweisung auf das obige Programmbeispiel angewendet und die Schleife bis zum Ende durchlaufen, so ergibt sich

BAD SUBSCRIPT ERROR IN 30

weil dort auf A(21) zugegriffen werden soll, da I nach dem Verlassen der Schleife den Wert 21 hat.

Vor dem Verlassen der Schleife müsste also die Ausgabe von A(I) wieder abgeschaltet werden:

.G,

Derselbe Fehler tritt auf, wenn Sie bei obigem Programm A(I+1) oder A(I-1) ausgeben lassen. Die Ausgabe von A(I+1) müssen Sie vor dem letzten Schleifendurchlauf abschalten. Entsprechend darf A(I-1) erst nach dem ersten Durchlauf ausgegeben werden, da sonst durch den negativen Index **ILLEGAL QUANTITY ERROR** auftreten würde.

- Reihenfolge der Ausgabe

Wenn die Variablenliste Variable oder Ausdrücke enthält, werden deren Inhalte oder Ergebnisse zuerst ausgegeben. Dann wird das nächste auszuführende Statement auf den Bildschirm gedruckt und dann ausgeführt.

Wenn Sie also ein einzelnes **.G** ausführen, erfahren Sie zuerst den bisherigen Inhalt der Variablen, und dann das nächste Statement, das auch gleichzeitig ausgeführt wird.



- Einzelschritt oder fortlaufender Trace

Wenn Sie die **RETURN** -Taste nur kurz antippen, wird nur das nächste Statement gelistet und ausgeführt und der Rechner meldet sich wieder mit **.G** zurück. Dadurch können Sie durch kurzes Antippen von **RETURN** ein Programm im Einzelschritt abarbeiten.

Solange Sie einfach **.G** durch **RETURN** übernehmen, ist die zuletzt definierte Variablentabelle gültig. Durch **.G,** wird die Variablenliste gelöscht. Durch Eingabe einer neuen Variablenliste wird ebenfalls eine vorher definierte vollständig gelöscht.

Durh ständiges Drücken von **RETURN** oder durch **SHIFT-LOCK** erreichen Sie einen kontinuierlichen Trace. Dieser Trace wird nur gestoppt, wenn Sie auf ein **INPUT** laufen.

- Tracen von GET

Tracen von **GET** (Tastatur) ist nicht möglich. Wenn **GET** im Programm vorkommt, müssen Sie durch Angabe von tracebeginn dafür sorgen, daß das Programm ohne Trace über die **GET**-Stelle laufen kann.

- Ausgabe des Trace auf beliebiges Peripheriegerät

Wenn Sie auf ein Peripheriegerät ausgeben wollen, für das **OPEN** erforderlich ist, dürfen Sie nicht durch **.R** starten, sondern müssen durch **.G erste zeile** starten, weil sonst die Datei wieder geschlossen wird.

Falls Sie den Trace auf Floppy ausgeben wollen und in dem zu tracenden Programm weitere **OPEN** auf Floppy vorkommen, müssen Sie darauf achten, daß weder die gleiche logische Adresse noch die gleiche Kanalnummer zweimal verwendet wird.

9. FLOPPY-BEDIENUNG

In Zelle 10pp steht als Voreinstellung Adresse 8. Durch **POKE 1000, gn** kann eine beliebige Gerätenümmer (gn) eingestellt werden.

9.1 Directory

Das Directory des Floppy kann durch

.@$ laufwerk:dateiname = typ

gelesen werden.

Beispiele:

.@$0	**Inhaltsverzeichnis** von Laufwerk 0
.@$1:DAT*	alle Dateien, die mit **DAT** beginnen
.@$1:*=P	alle **PRG**-Dateien
.@$1:DAT*=S	alle **SEQ**-Dat., die mit **DAT** beginnen

Die Ausgabe kann durch Loslassen der **RETURN**-Taste jederzeit unterbrochen werden. Durch die **STOP**-Taste kann **DIRECTORY** abgebrochen werden.

Die Liste von **DIRECTORY** kann nicht direkt auf Drucker gegeben werden, sondern erst hinterher durch **HARDCOPY,** soweit sie auf den Bildschirm gepaßt hat. Hierbei können Sie unmittelbar auf dem Bildschirm Kommentare hinter die Dateinamen schreiben und so das Inhaltsverzeichnis aussagekräftiger machen.

9.2 Floppy-Kommando

Die Floppy-Kommandos werden durch **´Klammeraffe´** eingeleitet (In der ersten Spalte muß natürlich noch der charakteristische **SM-KIT-Punkt** stehen).

Der Kommandostring kann auch Variable enthalten, wodurch z.B. **COPY** oder **SCRATCH** erleichtert wird. Deswegen müssen aber auch Textkonstante in Anführungszeichen eingeschlossen werden. Bei den Floppy-Kommandos gilt also die normale String-Syntax von BASIC und nicht irgendeine modifizierte Weise der Stringdarstellung.

Beispiele:

.@"C0:NEUDAT=1:ALTDAT

ALTDAT von Laufwerk 1 als **NEUDAT auf** Laufwerk 0 kopieren

N$="DATEI

.@"C0:"+N$+"=1:"+N$
DATEI von Laufwerk 1 auf 0 kopieren

- **Fehlermeldung lesen**

Ein **".@"** bringt den Floppy-Status (Floppy-Fehlermeldung)

10. LOAD

.L kann wie die BASIC-Anweisung **LOAD BASIC**-oder Maschinenprogramme laden. Zusätzlich kann aber durch Angabe einer Zeilennummer hinter dem Programmnamen ein **APPEND** durchgeführt werden. Wird hinter dem Programmnamen durch A eine Speicheradresse angegeben, wird die PRG-Datei ab dieser Adresse in den Arbeitsspeicher geladen. Wenn hinter dem Programmnamen ein Strichpunkt folgt, werden die Variablenzeiger nicht geändert, das BASIC-Programm und bestehende Variablen werden also durch Nachladen eines Maschinenprogramms nicht unbedingt verändert.

10.1 Formate

10.1.1 .L "programmname

.L hat gegenüber **LOAD** und **DLOAD** den Vorteil, daß keine Parameter angegeben werden müssen, da Gerät 8 fest eingestellt ist und das Floppy automatisch auf beiden Laufwerken sucht. Weiterhin kann hinter dem Programmnamen alles außer Strichpunkt, A oder einer Nummer folgen, ohne daß eine Fehlermeldung kommt oder die Funktion verändert wird.

Da man häufig vor dem Laden eines Programms das Inhaltsverzeichnis der entsprechenden Diskette auf den Bildschirm ausgeben läßt und dann dort vor den gewüschten Programmnamen **LOAD** schreibt, ist es ärgerlich, daß immer die Typenbezeichnung weggelöscht werden muß, um die **LOAD**-Syntax zufriedenzustellen.

Beispiel

16 "SM-KIT8-B7.1F" PRG

Hier schreiben Sie einfach in die erste Spalte den **Punkt** ,in die zweite L und löschen die dritte durch **SPACE.** Nach **RETURN** wird ohne **SYNTAX ERROR** geladen:

.L "SM-KIT8-B7.1F" PRG

10.1.2 .L "programmname", zeilennummer (APPEND)

Wird hinter dem Programmnamen durch Komma getrennt eine Zeilen-Nummer angegeben, so wird das Programm im Speicher ab dieser Zeile durch das angegebene Programm auf Diskette überschrieben. Das Diskettenprogramm wird also ans Speicherprogramm angehängt. Damit das so entstandene Programm lauffähig ist, müssen die Zeilennummern des Diskettenprogramms größer sein, als die angegebene Zeilennummer.

Beispiel:

Im Speicher steht ein Programm mit Zeilennummern von 10 bis 500 und auf Diskette steht ein Programm mit Nummern von 1000 bis 5000. Nach

.L"DISKPROG",1000

ist das Diskprogramm an das Speicherprogramm angehängt und kann so laufen. Nach

.L"DISKPROG",300

würde das Speicherprogramm ab Zeile 300 durch das Diskprogramm ersetzt. Das Ergebnis ist also wie vorher, aber die Zeilen 300 bis 500 des Speicherprogramms sind nicht mehr vorhanden.

Würden Sie aber die Zeilen von 1000 bis 5000 im Speicher stehen haben und das Programm mit 100-500 durch

.L"SPEIPROG",10000

anhängen, so erhalten Sie ein nicht-lauffähiges Programm, das am Anfang Zeilennummern von 1000-5000 hat und am Ende 100-500!

Beachten Sie, daß die Funktion **MERGE** die beiden Programme aber völlig einwandfrei zusammengeladen hätte.

Am Anfang kann es vorkommen, daß Sie aus alter Gewohnheit ,8 ans Ende von **.L** schreiben und damit die Gerätenummer meinen. SM-KIT meint aber in diesem Fall, daß Sie ab 8 anhängen/überschreiben wollen!

10.1.3. .L "name",A adresse

Durch Angabe einer Speicheradresse hinter **"name",A** laden Sie den Inhalt einer PRG-Datei ab der angegebenen Adresse in den Arbeitsspeicher.

10.1.4. LOAD + RUN

Durch **´Pfeil nach oben´** können Sie ein Programm laden und sofort starten:

.↑"PROGR" lädt und startet das Programm **PROGR**

11. MERGE

MERGE mischt beliebige Zeilen eines Diskettenprogramms in ein Speicherprogramm. Dabei existieren keinerlei Beschränkungen bezüglich Zeilennummern und Zeilenreihenfolge. Die Übernahmelogik entspricht der manuellen Übernahme vom Bildschirm, d.h. jede gefundene Zeile wird genau dort eingefügt, wo sie ihrer Nummer nach hingehört. Bereits vorhandene Zeilen werden durch die neue Zeile ersetzt.

11.1 Format

.M "programmname", erstezeile - letztezeile

Programmname und Zeilennummern sind als Konstante anzugeben, Variablen sind nicht zugelassen.

Der Zeilenbereich wird in derselben Weise wie bei **LIST** angegeben und kann insbesondere auch entfallen, dann wird das ganze Programm geladen.

11.2 Beispiel:

Sie haben ein Programm im Speicher mit den Zeilen 10, 20 und 30 und ein Programm auf Diskette mit den Zeilen 5, 15, 20 und 25. Mit diesen beiden Programmen werden in den folgenden Beispielen alle Möglichkeiten durchgespielt, die **MERGE** bietet.

P2 laden und listen (zuerst werden beide Programme vorgestellt)

```
.L"P2
LIST

   5 REM P2 -  5
   15 REM P2 - 15
   20 REM P2 - 20
   35 REM P2 - 35
   READY.
```

P1 laden und listen:

```
.L"P1
LIST

   10 REM P1 - 10
   20 REM P1 - 20
   30 REM P1 - 30
   READY.
```

P2 mergen:

```
.M"P2
LIST

   5 REM P2 -  5
   10 REM P1 - 10
   15 REM P2 - 15
   20 REM P2 - 20
   30 REM P1 - 30
   35 REM P2 - 35
   READY.
```

Die Zeilen von P2 wurden richtig in P1 eingebaut



P1 laden:

.L"P1

Zeile 5 von P2 einbauen (Zeile 5 ist stellvertretend für Zeilen, die **vor** dem Speicherprogramm eingefügt werden).

.M"P2",5
LIST

```
5 REM P2 -  5
10 REM P1 - 10
20 REM P1 - 20
30 REM P1 - 30
READY.
```

Zeile 15 von P2 einbauen (Zeile 15 ist stellvertretend für Zeilen, die **zwischen** andere Zeilen des Speicherprogramms eingefügt werden).

.M"P2",15
LIST

```
5 REM P2 -  5
10 REM P1 - 10
15 REM P2 - 15
20 REM P1 - 20
30 REM P1 - 30
READY.
```

Zeile 20 von P1 durch Zeile 20 von P2 ersetzen. (Zeile 20 ist stellvertretend für Zeilen, die bereits existieren und deshalb ersetzt (gelöscht) werden.

```
.M"P2",20
LIST

      5 REM P2 -  5
      10 REM P1 - 10
      15 REM P2 - 15
      20 REM P2 - 20
      30 REM P1 - 30
      READY.
```

Zeile 35 von P2 einbauen. (Zeile 35 ist stellvertretend für Zeilen, die ans Programmende angehängt werden).

```
.M"P2",35
LIST

      5 REM P2 -  5
      10 REM P1 - 10
      15 REM P2 - 15
      20 REM P2 - 20
      30 REM P1 - 30
      35 REM P2 - 35
      READY.
```

12. SAVE+VERIFY

Durch .S ... wird ein Programm abgespeichert und automatisch verifiziert. Das Programm kann ein BASIC-Programm sein, ein Teil eines BASIC-Programms oder ein Maschinenprogramm bzw. der Inhalt eines Arbeitsspeicherbereiches.

Durch das automatische **VERIFY** wird entweder **OK** gemeldet oder je nach aufgetretenem Fehler **FILE NOT FOUND ERROR** oder **VERIFY ERROR.**

12.1 Abspeichern eines BASIC-Programms

.S"laufwerk:programmname"

Beispiel:

.S"1:PROGRAMM1

speichert **Programm 1** auf Laufwerk 1 und meldet **OK,** wenn die Abspeicherung richtig durchgeführt wurde.

12.2 Abspeichern eines Zeilenbereiches

.S"laufwerk:programmname" , anfangszeile - endzeile

anfangszeile und endzeile wird wie bei **LIST** angegeben.

Abgespeicherte Zeilenbereiche können nur durch **MERGE (.M)** oder **APPEND (.L"name",zeile)** geladen werden!

Beispiel

.S"1:PROG",50-200

speichert die Zeilen von 50 bis 200 auf Laufwerk 1.

12.3 Adressbereich abspeichern

Durch **.S** können auch beliebige Speicherbereiche abgespeichert werden. Diese Funktion hat die gleiche Wirkung wie **.S** des Monitors, der Adressbereich wird aber dezimal angegeben und die Einschränkung der Namenlänge auf 15 Zeichen existiert nicht.

.S"laufwerk:programmname" , A anfangsadr. - endadr.+1

Wie bei **.S** des Monitors wird ab einschließlich der angegebenen Anfangsadresse bis ausschließlich der Endadresse abgespeichert.

Beispiel:

.S"0:MASCH1",A32768-34768

speichert den Bildschirminhalt als **MASCH1** auf Laufwerk 0.



A. ANHANG

A1. Wie ist SM-KIT ins Betriebssystem eingebaut?

SM-KIT verändert den Monitor-Zeiger in **1018/1019** beim Einschalten und restauriert die **RESET**-Werte beim Ausschalten mit **".X"**.

SM-KIT verändert die Zeiger für **BRK** (Software-Interrupt) in 146/147 und für **IRQ** (Maskierbarer Hardwareinterrupt) in 144/145 beim Einschalten und restauriert die **RESET**-Werte beim Ausschalten mit **".X"**.

Bei eingeschaltetem SM-KIT wird ständig kurzfristig die **CHR-GET** Routine in 112 verändert, indem bei 112 ein **JMP** auf SM-KIT eingebaut wird. Diese Umleitung wird aber dann sofort wieder entfernt. Etwaige Umleitungen, die weiter hinten in der **CHR-GET/GOT**-Routine stehen, werden nicht beeinflußt und müssten funktionsfähig bleiben.

A2. Welche Speicherbereiche werden verwendet?

Das SM-KIT-Programm belegt im **ROM**-Bereich den 4-K-Block **von 36864 ($9000) bis 40959 ($9FFF).**

Im **RAM** werden Zellen der Zero-Page und des zweiten Kassettenpuffers verwendet.

Zero-Page-Zellen, die vom Betriebssystem verwendet werden, werden vom SM-KIT in der Regel in der gleichen Weise verwendet wie vom Betriebssystem.

Zero-Page-Zellen, die vom Betriebssystem nicht verwendet werden, haben beliebige Funktionen, werden aber nur zeitlich lokal verwendet. SM-KIT rettet solche Zellen nicht, setzt aber auch nicht voraus, daß seine Zellen unverändert bleiben. Hier kann es also zu Interferenzen mit betriebssystem-fremden Maschinenprogrammen kommen.

Im zweiten Kassettenpuffer werden fast alle Zellen verwendet, die vom DISK-BASIC bzw. vom Tabulator nicht verwendet werden. Darunter befinden sich Adressen, deren Inhalt von fremden Programmen nicht verändert werden darf, wenn SM-KIT funktionsfähig bleiben soll.



A3. Welche Adressen werden verwendet?

Adresse	lesend	schreibend	Adresse	lesend	schreib.
3	*	*	4	*	*
5	*		7		*
9		*	17	*	*
18	*	*	31	*	
32	*		33	*	*
34	*	*	35	*	*
37	*	*	38	*	*
39	*	*	40		*
41		*	42	*	*
43	*	*	44		*
45		*	46		*
47		*	48		*
49	*	*	50	*	*
51	*		53	*	
54		*	55	*	*
56	*		57	*	
58	*	*	59	*	*
66	*	*	67	*	*
74	*		84	*	*
87	*	*	88	*	*
92	*	*	93	*	*
95	*		96	*	
97		*	98		*
112	*	*	113	*	
114	*		119	*	*
120	*	*	136	*	*
137	*	*	138	*	*
139	*	*	140	*	*
150	*	*	151		*
152		*	155	*	
156	*	*	157	*	
158	*	*	163	*	
164		*	167	*	
168	*		171		*
173	*		176	*	

Adresse	lesend	schreibend	Adresse	lesend	schreib.
177	*	*	178	*	*
179	*	*	180	*	*
181	*	*	182	*	*
183	*	*	184	*	*
185	*	*	186	*	*
187	*	*	188	*	*
189	*	*	190	*	*
191	*		192	*	
193	*	*	194	*	*
195	*	*	198	*	*
199	*	*	200	*	*
201	*	*	202	*	
211	*	*	212	*	
213		*	216	*	*
217	*	*	223	*	*
251	*	*	252	*	*
253	*		254	*	*

Die Adressen **512 bis 591** und **919 bis 1000** werden lesend und schreibend verwendet, die Adressen **622 bis 631** nur lesend.

A5 Fehleranhang für SM-KIT-17.02.81

1. Drucker ohne AUTO-LINE-FEED

Drucker ohne **AUTO-LINE-FEED** Möglichkeit können vom SM-KIT nicht bedient werden, da er nur **CR=CHR$(13)** sendet, nicht aber zusätzlich **LF=CHR$(10).**

2. MERGE / DOS-SUPPORT

Zumindest die BASIC-4 Version des **DOS-SUPPORT** Hilfsprogramms **(UNIVERSAL WEDGE)** von Commodore hat einen Fehler, der in Zusammenhang mit **MERGE** böse Auswirkungen zeigt:

Beim Laden von Programmen mit '/' oder ' ' hat **DOS-SUPPORT** den Programm-Ende-Zeiger (42/43) um ein Byte zu hoch gesetzt. Hat man ein so geladenes Programm wieder abgespeichert, so wurde am Ende nach den drei Nullen dieses Byte zusätzlich mit abgespeichert und beim nächsten Mal mit reingeladen. Durch mehrmaliges Wiederholen dieses Vorgangs entsteht nach dem Programmende ein Byte-Friedhof mit völlig zufälligem Inhalt. Soll **MERGE** - nun ein solches Programm bis zum Ende laden, kann das Programm im Speicher einer beliebigen Zeile abgeschnitten, also unvollständig sein, obwohl es vollständig geladen wurde.

Tritt dieser Fehler auf, so können Sie ihn umgehen, indem Sie das zu ladenende Programm nicht bis zum Ende, sondern nur bis zur vorletzten Zeile mit **MERGE** laden und die letzte nochmal von Hand eingeben.

3. AUTO

AUTO berechnet die nächste Zeilennummer aus der Differenz zur letzten. Wenn die daraus berechnete Zeilennummer größer als 65535 wäre, wird als neue Zeilennummer die theoretisch berechnete - 65536 ausgegeben, also eine kleinere Zeilennummer, als die letzte!



4. Dump von Feldinhalten

Wenn Felder verwendet werden, deren **letzte** Dimension mehr als 255 Elemente hat, wirkt bei gedrücktem **RETURN** oder **SHIFT-LOCK** die **Dauerfunktion nicht.**

Bei **DIM A(1,300)** oder **DIM B(300)** tritt dieser Fehler also auf, bei **DIM C(300,1)** nicht.

5. Trace von Gleitkommaoperationen

Bei allen Gleitkommaoperationen, die intern eine Division auslösen, funktioniert die Einzelschrittabfrage nicht, so daß Trace diese Anweisung und die folgende auf jeden Fall ausführt, ohne daß gestoppt wird, obwohl **RETURN** nicht gedrückt ist:

```
10 A = 3.7
20 A = A/2
30 GOTO 20
```

Wenn Sie dieses Programm mit **.r** starten und **RETURN** sofort loslassen, kommt der Trace erst in Zeile 30 mit **.g** zurück.



Notizen:



Notizen:

SM Softwareverbund-
Microcomputer GmbH
Scherbaumstraße 29
8000 München 83

Telefon: 089/6 37 12 11
Aut. Auftragsannahme
Telefon: 089/6 70 58 13